I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1757

GENERIC APPLICATION DESCRIPTION MODEL: TOWARD AUTOMATIC DEPLOYMENT OF APPLICATIONS ON COMPUTATIONAL GRIDS

SÉBASTIEN LACOUR, CHRISTIAN PÉREZ, THIERRY PRIOL

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Generic Application Description Model: Toward Automatic Deployment of Applications on Computational Grids

Sébastien Lacour, Christian Pérez, Thierry Priol*

Systèmes numériques
Projet Paris

Publication interne n1757 — October 23rd, 2005 — 21 pages

Abstract: Computational grids promise to deliver a huge computer power as transparently as the electric power grid supplies electricity. Thus, applications need to be automatically deployed on computational grids. However, various types of applications may be run on a grid (component-based, MPI, *etc.*), so it may not be wise to design an automatic deployment tool for each specific programming model. This paper promotes a generic application description model which can express several specific application descriptions. Translating a specific application description into our generic description is a simple task. Then, developing new planning algorithms and re-using them for different application types will be much easier. Moreover, our generic description model allows to deploy applications based on a programming model combining several models, as parallel components encompass component-based and parallel programming models for instance. Our generic description model is implemented in an automatic deployment tool which can deploy CCM and MPICH-G2 applications.

Key-words: Application Description, Deployment Planning, Automatic Application Deployment, Component, MPI, Computational Grids.

(Résumé : tsvp)

This paper is an extended version of a short paper published in the Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid2005), Seattle, WA, USA, November 2005, held in conjunction with SuperComputing 2005.

* {Sebastien.Lacour,Christian.Perez,Thierry.Priol}@irisa.fr

Centre National de la Recherche Scientifique (UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique et en Automatique – unité de recherche de Rennes

Modèle de description générique d'applications pour l'automatisation du déploiement d'applications sur des grilles de calcul

Résumé : Les grilles de calcul visent à offrir une puissance de calcul gigantesque de manière aussi transparente que le réseau de distribution électrique fournit l'électricité. Ainsi, les applications doivent pouvoir se déployer aussi automatiquement que possible dans les environnements de grilles de calcul. Cependant, divers types d'applications sont susceptibles de s'exécuter sur une grille (composants logiciels, MPI, *etc.*). Il ne semble donc pas approprié de concevoir autant d'outils de déploiement automatique qu'il existe de modèles d'applications. Ce papier présente un modèle de description générique d'applications qui permet de décrire dans un formalisme unique des applications de types variés. La conversion d'une description spécifique d'application en une description générique est une opération relativement simple. La notion de description générique d'applications permet à un unique planificateur de déploiement de placer des applications de types variés sur les ressources des grilles de calcul. De plus, notre modèle de description générique permet de planifier le déploiement d'applications fondées sur une combinaison de modèles de programmations, telles que les composants parallèles. Notre modèle de description générique d'applications est implémenté dans un outil de déploiement qui permet de lancer automatiquement des applications CCM et MPICH-G2.

Mots clés : description d'applications, planification de déploiement, déploiement automatique d'applications, composants, MPI, grilles de calcul.

1 Introduction

One of the long-term goals of computational grids is to provide a vast computer power in the same way as the electric power grid supplies electricity [13], *i.e.* transparently. Here, *transparency* means that the user does not know what particular resources provide electric or computer power. So the user should just have to submit his or her application to a computational grid and get back the result of the application without worrying about resource types, location, and selection, or mapping processes on resources. In other words, application deployment should be as *automatic* and easy as plugging an electric device into an electric outlet.

Many programming models are being developed to build grid applications, ranging from MPICH-G2[1] [18] to component-based models [30]. If we restrict ourselves to compute-intensive applications (*e.g.*, multi-physics simulations) for which computational grids appear very promising, several models are available, like CCA [4], Assist [1], ICENI [15], or GRIDCCM [26]. Most of those models combine a parallel technology (*e.g.*, MPI [17]) with a component-based technology which may be distributed like Web Services or CORBA. So not only are there many models, but there also exist models which are made up of several models. Hence, applications built using a combined programming model are inherently complex to deploy.

In addition, the nature of a grid environment makes it even more difficult to deploy an application: a large number of resources must be selected, the resources of a grid are fairly heterogeneous, various submission methods may be used to launch processes remotely, security must be enforced, *etc.*

Hence, our objective is to move toward the usage transparency targeted by computational grids, but not reached yet. We pursue this objective by building a tool designed for automatic deployment of distributed and parallel applications. This goal differs from such projects as NetSolve [5], APST [9], and Condor [31] which assign a *unique program* (possibly parallel) to one or more execution servers. In contrast, we need to run applications made of *multiple, communicating programs* (like multi-physics code coupling) on *several distributed resources*. In addition, contrarily to NetSolve and DIET [12], we do not assume that a specific service daemon is running on the execution servers of the grid. We cannot make this assumption because grid users may be interested in running a wide range of applications, and the resources of a grid are not devoted to a particular type of application: so we cannot assume specific daemons are running on each grid node for every particular type of application. The only assumption we make is that a grid access middleware is available on the grid resources, like the Globus Toolkit [16] for example.

The central element of automatic application deployment is the planner, which places the different parts (*e.g.*, processes, or programs) of the application on the various computers of the grid automatically. It usually requires a complete description of the application and available resources in input. As each type of application has its own description format, there are as many deployment planners as types of applications *a priori*. Not only does this

[1] MPICH-G2 is a grid-enabled MPI implementation relying on the Globus Toolkit version 2.

lead to a duplication of effort, but it is also an obstacle to building applications based on several technologies, like parallel components. Hence, it would be helpful to transform *specific* application descriptions into a *generic* application description to feed the planner.

This paper proposes an approach to make it easier to deploy a range of applications, focusing on a generic application description model. The generic application description format is independent of the nature of the application (*i.e.*, distributed or parallel), but complete enough to be exploited by a deployment planning algorithm. This application description model is integrated in the automatic application deployment framework which we have already proposed earlier [23]: it supports distributed component-based applications, parallel MPI applications, as well as parallel component-based GRIDCCM applications.

This paper is organized as follows. Section 2 presents a few types of application descriptions, in particular for component-based and MPI applications. Section 3 reviews the technologies currently used to deploy a grid application, and Section 4 presents our automatic deployment architecture. The contribution of this paper, a generic application description model, is presented in Section 5, and an example of utilization with our automatic deployment tool is given in Section 6. Section 7 concludes this paper.

2 Various Types of Applications to Deploy

Computational grids are not restricted to a particular class of applications. Users may wish to run distributed applications, parallel applications, or a combination of both on a computational grid. This section illustrates a few examples of application classes which we need to be able to deploy on a grid. It also shows how component-based and MPI applications may be described and packaged for automatic deployment.

2.1 Component-Based Applications

The component-based programming model makes it simpler to build distributed applications by *assembling components* rather than *writing* source code. A component-based application is made of a set of components interconnected through well-defined ports. As an example, we consider the CORBA Component Model (CCM [25]) because it specifies a packaging and deployment model. However, the following holds for any component-based application in general, like the Common Component Architecture (CCA [4]).

A CCM application package contains one or more components as well as two specific pieces of information. First, component placement information may require component collocation within the same host (to share physical memory for instance), or within the same process (to share a common address space for instance). Second, component interconnection information describes what component ports should be connected to other components' ports.

A CCM component package description enumerates the different compiled implementations of the component included in the package along with their target operating systems

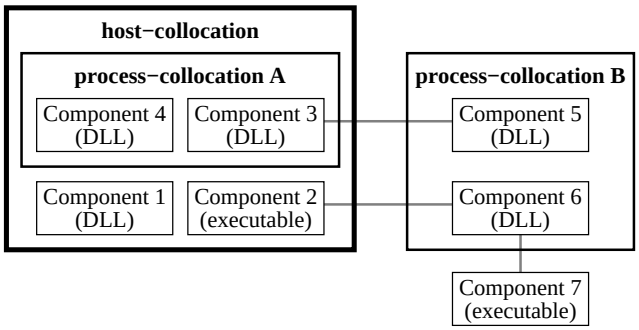

Figure 1: Example of CCM component-based application.

and computer architectures. Figure 1 shows an example of CCM application consisting of seven components made of executables and DLLs, and requiring host and process collocation.

2.2 MPI Applications

MPI (Message-Passing Interface) is an API to write parallel applications. An MPI application is made of only one program (SPMD) possibly compiled for various architectures and/or operating systems, and it is implicit that MPI processes connect to each other by themselves. The degree of parallelism is a specific piece of information of MPI applications: the description of an MPI application may determine the (initial) number of MPI processes to launch.

The only attempt to specify how MPI applications may be described and packaged which we are aware of is our previous work [24]. Figure 2 shows an example of how an MPICH-G2 application may be described. In that example, the application must be run with 32 processes. Two implementations of the MPI program are available and may be combined at runtime: one for i386 machines under Linux, and one for Sparc machines under Solaris. MPI applications may be packaged as CCM applications, in a ZIP compressed archive: the executables may be located within the archive, or remotely using a URL.

2.3 Parallel Component-Based Applications

Parallel component-based applications are made of components interconnected through well-defined ports too, but some components may be parallel programs, like MPI programs, instead of being sequential. There are many works in that area: XCAT and Ccaffeine are two frameworks of the Common Component Architecture (CCA). While XCAT [19] deals with distributed components, Ccaffeine [2] supports SPMD components. ICENI and

```
<MPI_appl type="MPICH-G2" count="32">
   <implementation id="1">
      <OS name="Linux"/>
      <ISA name="i386"/>
      <location type="local">
         appl.exe
      </location>
   </implementation>

   <implementation id="2">
      <OS name="Solaris"/>
      <ISA name="Sparc"/>
      <location type="URI">
         http://mpi.store.org/FFT.exe
      </location>
   </implementation>
</MPI_application>
```

Figure 2: Example of MPI application description.

ProActive [6] are two distributed JAVA component-based frameworks which support MPI-based parallel components (ProActive is based on Fractal and provides a hierarchical component model). Assist [1] is a component-based, parallel, structured programming framework mostly written in C++. Last, GRIDCCM is our parallel extension to the CORBA Component Model (CCM): it is implemented in C++ and supports MPI.

The description of applications pertaining to any of those models includes specific information related to both component-based and MPI applications.

3 How Applications Get Deployed Currently

This section illustrates how the application types mentioned in the previous section may be deployed on a computational grid. The objective is to show that manual deployment is too complex and requires too much expertise from the user.

3.1 Unicore and the Globus Toolkit

Unicore and the Globus Toolkit are grid access middleware systems: they make it easier for a user to securely launch various programs on remote, heterogeneous resources using a uniform interface. However, the user must still *manually* analyze the application, split it into parts, select execution resources, and place the application parts on the selected compute resources.

```
(&(resourceManagerContact="clstr.site1.org")
  (count=4)
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
           (GLOBUS_LAN_ID Univ_of_Stanford)
       (LD_LIBRARY_PATH "/usr/globus/lib"))
  (executable="/home/user/my_MPI_app_i386")
  (directory="/home/user/"))
(&(resourceManagerContact="clstr.site2.org")
  (count=8)
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
    (GLOBUS_LAN_ID Univ_of_Stanford)
    (LD_LIBRARY_PATH "/ap/globus2.4.3/lib"))
  (executable="/home/john/MPI_appl_sparc"))
```

Figure 3: Example of RSL script to launch an MPICH-G2 application (simplified).

For instance, the Globus Resource Specification Language (RSL [28]) is used to deploy MPICH-G2 applications among others: as shown on Figure 3, an RSL script to deploy an MPICH-G2 application is not an application description, since it mixes information related to the application (location of the executables, number of instances of a program, *etc.*) with information pertaining to execution resources (names of computers, job submission method, *etc.*) as well as information on the execution environment (working directory, environment variables, *etc.*) That information combination makes it more difficult to deploy the same application in another environment, should it be another grid, or a different system like a cluster. Before writing such an RSL script, the user must discover the available grid resources by himself (using any grid resource information service, like the Globus MDS[2] [10]), find their characteristics (operating systems, computer architectures, *etc.*), and place each executable of the application on the resources. That is too complex for grid users and distracts them from their scientific focus, so the deployment process should be automated.

3.2 Condor-G and GridLab GAT/GRMS

Condor-G [14] and GridLab's GAT[3] [3] manage the whole process of remote job submission to various queuing systems (*e.g.*, PBS, LSF, Sun Grid Engine, *etc.*) They provide a matchmaking [27] feature to place executables on resources, and launch those executables on the selected remote resources. Both can deploy MPI applications, but those parallel applications must be deployed on a *single* and *uniform* cluster.

[2]Monitoring and Discovery Service.
[3]Grid Application Toolkit.

Condor's ClassAd mechanism does not allow for an MPI application to span multiple clusters (should they be homogeneous or not). GridLab GAT/GRMS[4] [20] and its associated "Job Description" can only describe applications made of a *unique* executable, while most code-coupling applications are composed of a number of programs (simulating various physics). GRMS Job Description also mixes information related to the application with information on execution resources and environment, while those pieces of information should be separated.

Last, there may be network requirements between the various, distributed parts of an application: CCM components have interconnection constraints (Section 2.1), and MPI applications may need certain communication characteristics between their processes [24]. However, Condor ClassAds and GridLab's GRMS cannot describe connection constraints between the various parts of an application.

4 Automatic Deployment Architecture Implemented in the deployment tool ADAGE

As seen in previous section, the deployment method depends on both the application type and the nature of the target execution environment. So the user must learn as many deployment methodologies as application types and execution environments. Moreover, existing deployment solutions do not account for complex applications, made of various parts distributed over the sites of a grid. To let the user really focus on science rather than computer technology, application deployment must be automated.

This section summarizes the general process to *automatically* deploy applications on a computational grid as we have already presented it in previous work [23].

4.1 Architecture Overview

As shown on Figure 4, a set of input files including the application description is turned into a deployment plan whose execution results in staging files in and launching processes. The deployment planner is at the heart of automatic application deployment: the planning phase consists in selecting compute nodes, and mapping the application parts onto the selected compute nodes. It is out the scope of this paper to deal with actual scheduling algorithms [7].

In input, the deployment planner needs a description of the application and a description of the available resources; it may also accept various control parameters.

The application description enumerates the various parts of the application to be deployed. We use the term "parts of application" because the nature of those parts depends on the application type: component-based applications are made of components, MPI applications are composed of processes, *etc.* Application description includes pointers to the

[4] Grid Resource Management Service.

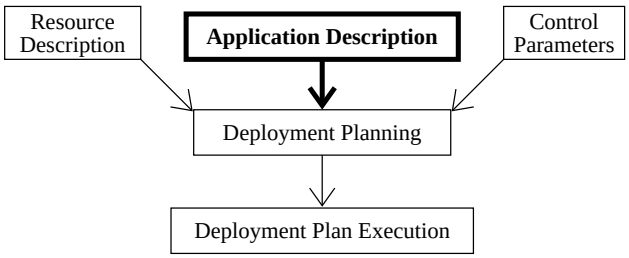

Figure 4: Overview of the general process of automatic application deployment. The focus of this paper is figured in bold.

binary files of the application's compiled implementations (*i.e.*, executables or DLLs), information on their target operating systems and computer architectures, their dependencies on libraries, *etc.*

The grid resource description describes storage and compute resources (CPU count, memory size, operating system, computer architecture, *etc.*), but also includes information on the network interconnections between compute nodes, like topology and performance characteristics. We have already proposed scalable model for grid network description in [22].

Control parameters are additional requirements which the user may impose to keep a certain level of control on the deployment process. The user may wish to minimize the execution time by selecting the clusters made of the computers with the highest CPU speeds available, or run the application at a particular site close to a visualization node, or minimize the total cost of the selected execution resources, *etc.*

The output of the planner is a deployment plan which specifies which part of the application will execute on which compute resource of the grid. Then the deployment plan is executed: various files (executables, DLLs, *etc.*) are uploaded and installed on the selected compute nodes, and processes are launched using the remote process creation method specified by the deployment plan (*e.g.*, SSH, Globus GRAM [11], *etc.*) In this automatic deployment architecture, middleware systems like the Globus Toolkit, Condor, Unicore, GridLab's GAT are useful to execute the deployment plan and launch processes on remote grid resources.

4.2 ADAGE: a Prototype Implementation

The automatic deployment architecture described above has been implemented in our prototype tool named ADAGE. Currently, it can deploy CCM applications as well as MPI applications, and it has two distinct planners for each type of application. ADAGE relies on the Globus Toolkit and SSH to launch processes on grid resources. Our practical experience

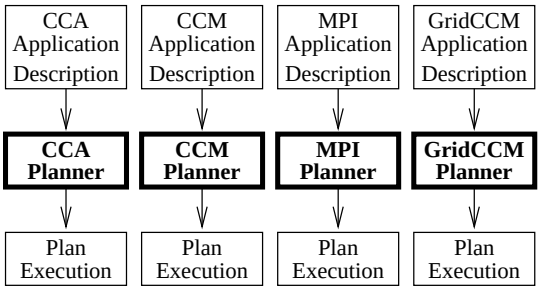

Figure 5: As many complex, application-specific deployment planners as types of applications.

on automatic deployment of component-based applications on computational grids using ADAGE is described in [21].

4.3 Discussion

To summarize, the proposed architecture enables automatic deployment of applications, as demonstrated by our prototype ADAGE. However, such an automatic deployment architecture may be implemented for as many types of applications which the user may want to run on a grid. As shown on Figure 5, implementing multiple deployment planners for various types of applications with the same planning algorithm may not be an adequate approach because that is a complex and tedious task. Moreover, advanced programming models based on both parallel and distributed technologies, like GRIDCCM, raise the issue of their deployment method, and implementing a specific planner is a duplication of effort since another planner for CCM and MPI must be written again. The following section proposes a solution to those issues.

5 GADE: a Generic Application Description Model

5.1 Motivations and Objectives

Section 2 showed that there are several ways to describe an application, depending on the nature of the application itself: component-based applications are described in terms of interconnected *components* possibly collocated, MPI applications are described in terms of *processes*. This application description diversity would suggest that different deployment planners would be implemented depending on the type of application to be deployed (Figure 5).

However, a planning algorithm is difficult to design and implement since it holds most of the intelligence of the deployment tool: every single decision is made by the deployment

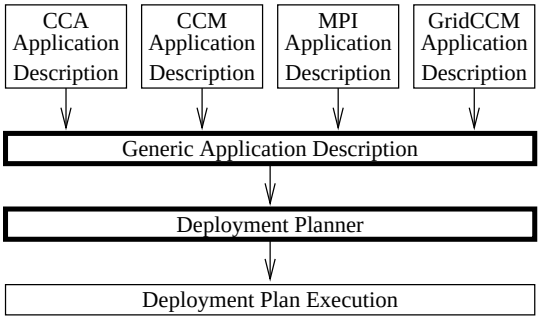

Figure 6: Conversion from specific application descriptions to a generic description format, and a unique deployment planner.

planner, and deployment plan execution involves no decision. So writing a planner for every type of application may be too costly.

Our objective is to capitalize on a deployment planning algorithm once a planner has been written to accept in input a generic application description. Hence, as shown on Figure 6, specific application descriptions are translated into generic application descriptions which are the input of the deployment planner. This is reasonable since all the applications we are interested in are similar in that they all end up as running threads and processes, possibly communicating with each other. Indeed, the deployment planner does not need to be aware that it places components or MPI processes on grid resources: it just assigns *computing entities* to compute nodes.

Last, different types of applications may be deployed on a grid, including applications based on a *combination* of parallel and distributed paradigms, like parallel components. Should such combined applications be deployed using planners and launch methods specific to MPI applications or specific to component-based applications? None, there is a need for a *unified deployment method*.

This section presents our proposed generic application description model, whose goal is to abstract from any particular type of application to deploy. That will allow to re-use the same planning algorithm for various types of applications without re-implementing another planner specific to another particular type of application.

5.2 Generic Application Description Model Overview

As shown on Figure 7, in our generic application description model (GADE), an application is made of a list of "computing entity" hierarchies (*system entities*, *processes*, and *codes to load*) along with a list of connections between "computing entities", meaning that they may communicate with other "computing entities". The hierarchy is based on memory sharing.

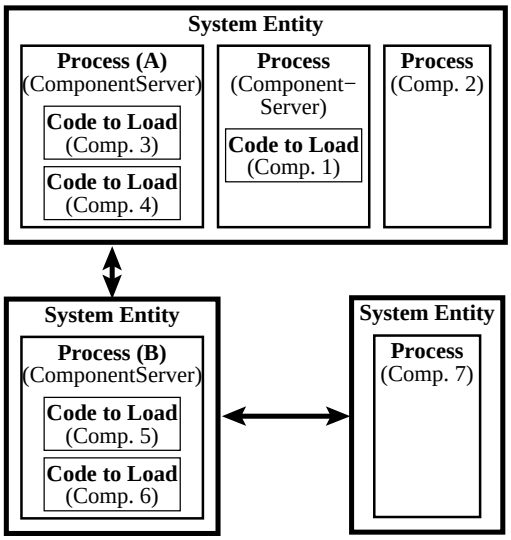

Figure 7: Generic description of our example CCM application represented on Figure 1.

The three "computing entities" which we consider are *system entities*, *processes*, and *codes to load*. A code to load is part of a process, and all codes of a process share a common address space. A process is a running instance of a program with a private memory address space: it is included in a system entity, and all processes of a system entity run on the same host, sharing a common physical memory. Each system entity may be deployed on distributed compute nodes. The following section presents a specification of the "computing entities" and connections.

5.3 Generic Application Description Specification

5.3.1 System entity

A system entity (*i.e.*, a set of processes to be deployed on the same compute node) has a cardinality C_s, meaning that C_s instances of the system entity must be deployed on distributed compute nodes. It is up to the planner to determine whether the C_s identical system entities will run on one compute node, or on different distributed nodes. A set of resource requirements may be attached to a system entity, specifying a list of operating systems and/or computer architectures onto which the system entity may be mapped.

5.3.2 Processes

A process has a cardinality C_p, meaning that C_p instances of the process must be deployed within the system entity. A process may also have a list of implementations for various operating systems and computer architectures. An environment may be attached to a process, including environment variables, a working directory, library dependencies, *etc.* Finally, a startup method is attached to every process, specifying how the process may be launched: a JAVA process will be started using a JVM, MPI processes will be created using `mpiexec`, plain executables will be started using a program loader, *etc.*

5.3.3 Codes to load

As system entities and processes, codes to load have a cardinality C_c. A DLL corresponding to a code to load may have a list of implementations for various operating systems and computer architectures. A loading method must also be attached to a code: in case the code to load is a CCM component, it will be loaded into the process using the CCM operation `install(id, JAR or DLL file)`.

5.3.4 Connections

The generic application description also includes a list of connections between system entities. A system entity is connected to another one if they both contain processes or codes to load which will need to communicate (*e.g.*, interconnected components, MPI processes, *etc.*) The connections are between system entities only, since we assume the processes within a system entity can always communicate with each other, as well as the codes within a process.

5.4 Specific to Generic Application Description Conversion

A translator is responsible for converting a specific application description to our generic description format. There is one translator for each type of application, but a translator is quite straightforward to write. Let us examine translators for MPI and component-based applications.

5.4.1 MPI applications

The description of an MPI application made of n processes translates to a system entity of cardinality $C_s = n$: the system entity is made of one process since there is usually no need for host collocation in MPI applications. For MPI applications, there is no need to load codes from external initiatives, so there is no code to load in the generic descriptor of an MPI application. Finally there is a connection from the unique system entity to itself, which means that every instance of the system entity replicated C_s times must be able to communicate with every other instance.

5.4.2 Component-based applications

The components required to be collocated on the same host result in multiple processes within the same system entity.

Figure 7 corresponds to the transformation of the example of Figure 1. The upper system entity is made of three processes located on the same host. The components required to be collocated in the same process result in multiple codes to load in the same process. Process B corresponds to process-collocation B: it is made of two codes to load corresponding to components 5 and 6. In CCM, a component may be a standalone executable: it results in a process holding no code to load in our generic application description (like component 7 represented by just one process in a system entity on Figure 7). A CCM component may also be a DLL or JAVA `.class` file: in this case, it results in a process (called "ComponentServer") holding a code to load which represents this component (DLL, JAVA `.class` file, *etc.*) The connections in the generic application description reflect the connections between the components.

Section 6 will provide information on how we have implemented a translator for CCM to GADE. However, GADE is also able to deal with other component models. For instance, CCA applications can be described using GADE. Let us consider two CCA frameworks with two different parallelism models: XCAT and Ccaffeine. XCAT [19] deals with components and services distributed over a grid. Hence, each XCAT component can be converted into a system entity of GADE. It is very similar to CCM except that XCAT does not provide host or process collocation constraints like CCM yet. Ccaffeine [2] is a CCA implementation which primarily targets SPMD, parallel components. Ccaffeine components are implicitly within the same process. Hence, those components are translated into codes to load which belong to the same process, hosted by a single system entity. Like in MPI, the cardinality of the system entity controls the process count.

5.5 Advantages over Specific Application Descriptors

The conversion from a specific application description to a generic description makes the application description independent of the nature of the application, but makes it more dependent on a computer model: our generic description model assumes a single operating system per compute node which can run one or more processes sharing physical memory, and processes which may load one or more codes sharing the same virtual address space. However this assumption is reasonable in that this computer model is extremely common.

In addition, by introducing a generic application description model, we make planners simpler to implement since they do not have to cope with semantics implied by particular application types. For instance, the planner does not need to be aware of particular application notions, like components or host-collocation constraints, since they are already digested by the translator. That amounts to moving a bit of intelligence from the planner up to the translator from specific to generic application description.

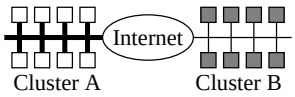

Figure 8: Example of grid resources available.

5.6 Impact on the Automatic Deployment Architecture

The generic description model does not contain enough information to configure the application after deployment plan execution: the original application description must be used again to configure the application and interconnect its various parts. For example, grid-enabled MPI libraries must be correctly set up with topology information. For CCM applications, configuration amounts to instantiating components, interconnecting them, setting the initial values of their attributes, *etc.* Hence, to support a type of application, a translator from the application representation to a GADE representation must be provided, as well as a plug-in to configure the application.

6 Utilization of GADE within ADAGE

This section deals with the implementation of GADE within ADAGE. It starts by describing an example, then it gives technical details on the translators which ADAGE supports.

6.1 Example of Utilization

Let us consider an application made of a parallel component (composed of four MPI processes) and a sequential visualization component. The grid resources consist of two clusters (Figure 8): cluster A is equipped with eight compute nodes with 1 GB of RAM each, and interconnected by a 2 Gb/s network; cluster B is made of eight nodes with 2 GB of RAM each, and interconnected by a 100 Mb/s network. The control parameters require that the four MPI processes of the parallel component be connected by a network of at least 1 Gb/s, and they specify that the visualization component must be run on a compute node with at least 2 GB of RAM. All that information is passed to the deployment tool.

First, the application description is translated into the generic description shown on Figure 9, whose textual representation is given in Figure 10, following the explanations of Section 5.4: the connection departing from and arriving to the system entity of cardinality $C_s = 4$ means that the planner must make sure every instance of the system entity can communicate with each other. Then, as shown on Figure 11, the planner produces a deployment plan which places the various parts of the application on the grid resources, checking the operating system and architecture compatibilities between computers and executables.

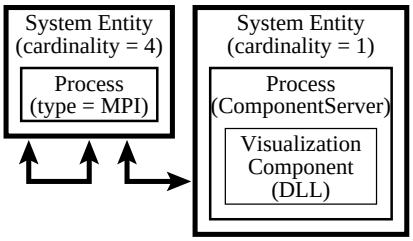

Figure 9: Schematic generic application description.

6.2 Implementation of Translators

We have modified ADAGE to support the generic application description so as to validate it. We have also taken this opportunity to move the internal C data structures to XML DOM representations. The main motivation was to benefit from a standard API, in order to be able to utilize XML Path (and later XML Query) as well as XSL transformation.

We have implemented three translators using three different approaches. For the CCM to GADE translation, we have written a translator in C language, since all the internal data structures were already represented in C (a plug-in of ADAGE can convert C data structures to a GADE representation). That represents about 1,200 lines of C. Then, ADAGE provides an operation to transform the C data structures into a DOM representation using Xerces-C++.

For MPI applications, we wrote a C++ plug-in which directly converts the XML representation of an MPI application to a DOM representation of GADE. It uses Pathan, a C++ XML Path implementation based on Xerces-C++, to perform lookups. It takes a little less than 300 lines of code. As explained in Section 5.4.1, the conversion is straightforward as MPI processes are converted into system entities.

Last, we are starting to use XSL to implement a translator from specific application description to GADE. We have developed a simple application representation for DIET [8], a grid computing environment based on the Grid-RPC approach [29]. This representation, which is a hierarchy of processes, is represented as a hierarchy of system entities in GADE. The conversion is actually done using Xalan-C++, an XSLT processor.

7 Conclusion

This paper presented an overview of the process of automatic application deployment in a grid environment, targeting more transparency in the utilization of computational grids. However, the diversity of types of applications susceptible of being deployed on a grid yields as many application description formats. This multiplicity of formats would result in as many specific planners. As a planner is difficult to design and implement, we do not

```
<generic_appl_descr>
   <system_entity id="se_0">
      <cardinality count="4"/>
      <process type="MPI">
         <binary>ftp://mpi.org/a.out</binary>
         <OS_name>Linux</OS_name>
         <ISA_name>x86_64</ISA_name>
      </process>
   </system_entity>

   <system_entity id="se_1">
      <cardinality count="1"/>
      <process type="plain">
         <binary>component-srvr.exe</binary>
         <OS_name>FreeBSD</OS_name>
         <ISA_name>i386</ISA_name>
         <code_to_load type="CCM">
            <DLL>visual.so</DLL>
         </code_to_load>
      </process>
   </system_entity>

   <connection type="self">
      <ref_to_sys_entity refid="se_0"/>
   </connection>

   <connection type="non_directed">
      <ref_to_sys_entity refid="se_0"/>
      <ref_to_sys_entity refid="se_1"/>
   </connection>
</generic_appl_descr>
```

Figure 10: Generic application description in XML.

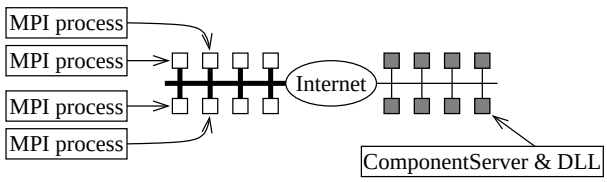

Figure 11: Deployment plan: selected compute resources are assigned with the various parts of the application.

wish to write as many planners as application description formats. Moreover, advanced programming models, like parallel components, are based on several programming models, like CCM and MPI for example. This raises the issue of how to deploy applications using such combined models. After identifying a few application-specific description elements, we proposed a generic application description model (GADE). Once a planner has been written to accept in input a generic application description, this model allows to capitalize on a planning algorithm in order to deploy a range of applications. It also solves the issue of advanced programming model as GADE is independent of any application-specific description. Finally, we illustrated how the generic application description model integrates in an automatic application deployment tool like our prototype: ADAGE is capable of automatically deploying both distributed CCM component-based applications and parallel MPICH-G2 applications on computational grids, using a unique planner. We also showed that the translators from specific to generic application descriptions are easy to write.

We are currently working on finalizing the prototype, in particular the DIET and GRID-CCM translators. A future step will be to understand how application-specific considerations should be handled for the configuration phase, once processes have been created remotely. Even though MPI and CCM rely on different APIs, the fundamental operations seem to be similar, *i.e.* connecting application parts and initializing parameters. A related question is how to efficiently execute a deployment plan. Currently, it is executed sequentially and in a centralized way, while it is a parallel and distributed operation.

References

[1] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. *Grid Computing: Software environments and Tools*, chapter ASSIST as a Research Framework for High-Performance Grid Programming Environments. Springer Verlag, 2004.

[2] Benjamin A. Allan, Robert C. Armstrong, Alicia P. Wolfe, Jaideep Ray, David E. Bernholdt, and James A. Kohl. The CCA core specification in a distributed memory SPMD framework. *Concurrency and Computation: Practice and Experience*, 14(5):323–345, 2002.

[3] Gabrielle Allen, Kelly Davis, Tom Goodale, Andrei Hutanu, Hartmut Kaiser, Thilo Kielmann, Andre Merzky, Rob van Nieuwpoort, Alexander Reinefeld, Florian Schintke, Thorsten Schütt, Ed Seidel, and Brygg Ullmer. The Grid Application Toolkit: Towards generic and easy application programming interfaces for the grid. In *Proc. of the IEEE*, volume 93, March 2005.

[4] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proc. of the 8th IEEE International Symp. on High Performance Distributed Computing (HPDC'99)*, pages 13–22, Redondo Beach, CA, August 1999.

[5] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users' guide to NetSolve v1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, Univ. of Tennessee, Knoxville, TN, June 2002.

[6] Francoise Baude, Denis Caromel, and Matthieu Morel. From distributed objects to hierarchical grid components. In *International Symp. on Distributed Objects and Applications (DOA)*, number 2888 in LNCS, pages 1226–1242, Catania, Italy, November 2003.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *Proc. of the 8th Heterogeneous Computing Workshop (HCW)*, pages 15–29, San Juan, Puerto Rico, April 1999.

[8] Eddy Caron, Frédéric Desprez, Frédéric Lombard, Jean-Marc Nicod, Martin Quinson, and Frédéric Suter. A scalable approach to network enabled servers. In B. Monien and R. Feldmann, editors, *Proc. of the 8th International Euro-Par Conference*, volume 2400 of *LNCS*, pages 907–910. Springer Verlag, August 2002.

[9] Henri Casanova, Graziano Obertelli, Francine Berman, and Rich Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the grid. In *Proc. of the IEEE/ACM SuperComputing Conference (SC'2000)*, pages 60–78, Dallas, TX, USA, November 2000.

[10] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid information services for distributed resource sharing. In *Proc. of the 10th IEEE International Symp. on High-Performance Distributed Computing (HPDC-10'01)*, pages 181–194, San Francisco, California, August 2001.

[11] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 62–82, 1998.

[12] DIET web site, `http://graal.ens-lyon.fr/DIET/`.

[13] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter Computational Grids, pages 15–51. Morgan Kaufmann, San Francisco, CA, January 1998.

[14] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proc. of the 10th International Symp. on High Performance Distributed Computing (HPDC)*, pages 55–63, San Francisco, CA, August 2001.

[15] Nathalie Furmento, Anthony Mayer, Stephen McGough, Steven Newhouse, Tony Field, and John Darlington. ICENI: Optimisation of component applications within a grid environment. *Journal of Parallel Computing*, 28(12):1753–1772, 2002.

[16] The Globus Alliance, `http://www.Globus.org/`.

[17] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 2nd edition, November 1999.

[18] Nicholas T. Karonis, Brian Toonen, and Ian Foster. MPICH-G2: a grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing (JPDC)*, 63(5):551–563, 2003.

[19] Sriram Krishnan and Dennis Gannon. XCAT3: A framework for CCA components as OGSA services. In *Proc. of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, pages 90–97, Santa Fe, NM, USA, April 2004.

[20] Krzysztof Kurowski, Bogdan Ludwiczak, Ariel Oleksiak, Tomasz Piontek, and Juliusz Pukacki. GRMS user guide version 1.9.6. Technical report, GridLab, Poznan Supercomputing and Networking Center, March 2005.

[21] Sébastien Lacour, Christian Pérez, and Thierry Priol. Deploying CORBA components on a computational grid: General principles and early experiments using the Globus Toolkit. In *Proc. of the 2nd International Working Conference on Component Deployment (CD 2004)*, number 3083 in LNCS, pages 35–49, Edinburgh, Scotland, UK, May 2004.

[22] Sébastien Lacour, Christian Pérez, and Thierry Priol. A network topology description model for grid application deployment. In *Proc. of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, pages 61–68, Pittsburgh, PA, USA, November 2004.

[23] Sébastien Lacour, Christian Pérez, and Thierry Priol. A software architecture for automatic deployment of CORBA components using grid technologies. In *Proc. of the 1st Francophone Conference On Software Deployment and (Re)Configuration (DECOR 2004)*, pages 187–192, France, October 2004.

[24] Sébastien Lacour, Christian Pérez, and Thierry Priol. Description and packaging of MPI applications for automatic deployment on computational grids. Research Report RR-5582, INRIA, IRISA, Rennes, France, May 2005.

[25] Open Management Group (OMG). CORBA components, version 3. Document formal/02-06-65, June 2002.

[26] Christian Pérez, Thierry Priol, and André Ribes. A parallel CORBA component model for numerical code coupling. *The International Journal of High Performance Computing Applications*, 17(4):417–429, 2003.

[27] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc. of the 7th IEEE International Symp. on High Performance Distributed Computing (HPDC7)*, pages 140–146, Chicago, IL, July 1998.

[28] Globus3 Resource Specification Language (RSL), `http://www-unix.Globus.org/toolkit/docs/3.2/gram/ws/developer/mjs_rsl_schema.html`.

[29] Keith Seymour, Craig Lee, Frédéric Desprez, Hidemoto Nakada, and Yoshio Tanaka. The end-user and middleware APIs for GridRPC. In *Proc. of the Workshop on Grid Application Programming Interfaces (GAPI'04)*, September 2004.

[30] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, first edition, 1998.

[31] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.